Exercises: Parallel merge and application to sort

Jean-Louis Roch

I. Complexity of MERGE and sequential algorithm

1. *There are C_{n+m}^n possible choices to fix n positions in X; the n distinct elements of A are then ranked on those fixed positions, in increasing order. Finally, the m distinct elements are ranked in increasing order in the m remaining positions of X.*

2. *Each comparizon may, in the best case, split the possible outputs in 2 half. Also, a lower bound is $\log_2 C_{n+m}^n$.*

3. *We have $\log_2 n! \simeq n\log_2 n - n\log_2 e + \frac{1}{2}\log_2 n + O(1)$ Thus a lower bound is $2n\log_2 2n - 2n\log_2 e - 2n\log_2 n + 2n\log_2 e - \log_2 n + \frac{1}{2}\log_2 n + O(1) \simeq 2n - frac12\log_2 n + O(1) \simeq 2n$.*

4.a. *In the first loop (the only one that includes comparizons), at each comparizon, an element is ranked. Since this loop ranks at most $n+m-1$ elements, it performs at most $n+m-1$ comparisons. Moreover, let consider the following instance: $a_0 < \ldots < a_{n-2} < b_0 < \ldots < b_{m-1} < a_{n-1}$: The $n-1$ first elements of A are compared to b_0 and the m elements of B are compared to a_{n-1}.*

4.b. *All comparizons in the first loop are in precedence (sequential dependency). In the general case, we have $D(n,m) \geq \min(n,m)$. In the previous instance, all comparizons are in precedence: thus $D(n,m) = n+m-1$ in the worst case.*

II. A parallel Divide&Conquer algorithm for MERGE

5.a. *Since A and B are sorted, A_1, A_2, B_1 and B_2 are sorted too. All elements of A_1 and B_1 are lesser than α, this than all elements in A_2 and B_2. Thus, we can merge independently the $n/2+j$ elements of A_1 and B_1 into the $n/2+j$ first positions of X; and the elements of A_2 and B_2 into the $n+m-n/2-j$ last positions of X. The resulting array is sorted.*
NB: if A or B are empty, it is sufficient to copy – in parallel – the elements of the other non empty array into X.

5.b. *We perform a dichotomic search in B : this requires $\lceil \log_2 m \rceil$ comparisons.*

5.c *The recurrence is direct; in worst case, m is maximum ($n=m$) and the array B is always on the same side so $D(n,n) \leq D(n/2,n) + \log n \leq D(n/2,n/2) + 2\log n - 1 = O(\log^2 n) = O(\log^2 n + m)$.*

5.d. $T_p < \frac{n+m}{p} + o(n+m) + O(\log^2 n + m)$.

III. An ultrafast parallel algorithm for MERGE

6.a. *The number of elements in X lesser than a_i is i in A and k in B, thus $i+k$.*

6.b. *For a fixed i, in parallel for all k: compute boolean $t_{i,k} := (b_{k-1} < a_i) \vee (b_k > a_i)$.*
Since all elements are distincts, only one $t_{i,k}$ is true. So the algorithm is :
parfor $k = 0..m-1$ { if $((b_{k-1} < a_i)$ and $(b_k > a_i))$ $k_i := k$ }.
This computation is of depth O(1) and performs m comparasons.

6.c. *The previous algorithm question 6.b is applied to all elements of A to compute their position in X in depth $O(1)$ with $n.m$ comparisons.*
• *In parallel for $i = 0..n-1$ doSEQ :*
1. *In parallel for $k = 0..m-1$ compute boolean $t_{i,k} := (b_{k-1} < a_i) \vee (b_k > a_i)$.*
2. *In parallel for $k = 0..m-1$: if $t_{i,k} == vrai$ then $x_{i+k} := a_i$.*

• *Proceed the same way for all elements of B.*
The two arrays A and B are thus merged in depth $O(1)$ with $2nm$ comparisons.

IV. An efficient cascading algorithm for MERGE

7.a. *Each μ_i can be ranked like in 6.a in depth 1 with $O(\sqrt{m})$ operations. This all the μ_i are computed in depth 1 with $O(\sqrt{n}.\sqrt{m}) = O(n+m)$ operations.*

7.b. *Once computed μ_i and μ_j, we have a partition of A (resp. B) in $\sqrt{n}+\sqrt{m}$ blocks, defined for A by the α_i and ν_j (resp. by the β_j and μ_i). Then, we have computed in depth $O(1)$ a sequence of $\sqrt{n}+\sqrt{m}$ couples of blocks (one for A, the other one for B); each couple corresponds to one block of A of size $< \sqrt{n}$ to merge recursively with one block of B with size $< \sqrt{m}$ éléments.*
We have $D(n) = D(\sqrt{n}) + O(1) = O(\log\log n)$ and $W_1(n) \leq \sqrt{n}.W_1(\sqrt{n}) + O(n) = O(n.\log\log n)$.

7.c. *Just split the array A (resp B) in $O(n/\log\log n)$ (resp. $O(m/\log\log m)$) blocks, and take the first elements of each block to build the array α (resp. β).*
Then, merge α and β ib depth $O(\log\log n)$ with $O(n)$ operations.
Finally, we have nos the list of couples to lerge; in each couple there are at most $\log\log n$ elements for A and at most $\log\log m$ elements for B. Thos small blocks with size $< \log\log n + m$ are merged sequentially. The algorithm has a depth $O(\log\log n)$ and each of the 3 phases performs $O(n)$ operations.

V. Application to parallel merge-sort

8. *We have $W_1(n) = 2W_1(N/2) + W_1^{(M)}(n)$ and $D(n) = D(N/2) + D^{(M)}(n)$. Solving the recurrence equations in the 3 cases leads to:*
9.a. *$D(n) = O(n)$ et $W_1^(n) = O(n\log n)$;*
9.b. *$D(n) = O(\log^3 n)$ et $W_1^(n) = O(n\log n)$;*
9.c. *$D(n) = O(\log n.\log\log n)$ et $W_1^(n) = O(n\log n)$;*